Monday 14 March 2016

An ideal(istic) approach on firmware developments in ATLAS.

S.Veneziano

The scope of this document is to find a way to have a lightweight procedure to follow firmware developments in ATLAS, starting from an ideal perspective.

Firmware projects needs to be integrated into current upgrade projects in a more structured way. This document describes an ideal methodology to develop large firmware projects, assuming that infinite manpower is available. Most of this text will be known by expert designers and considered just good design practice, but the audience of this document might be more extended, including people with experience in managing large projects of other kinds who wish to get an insight of firmware projects.

For such projects we will first need to define their global functionality, their main building blocks, target which blocks are common and which can be shared between projects, then define their interfaces.

In order to have many groups working on the same project we may need to provide for all building blocks interface specifications, behavioural models, eventually in a high-level language that may be used also for bit-wise simulations and for early integration of the blocks in a full project simulation.

We will also need to understand the physical implementation, like the resource usage of the proposed devices for each individual project and power consumption, so that we do not have surprises later on.

It may be too early now, not appropriate for PhaseI where we have embedded
processors only for control and configurations, but we will probably need to integrate standard processors in the trigger path, think of the muon trigger for PhaseII for example,
and we need to have tools (and languages) to do co-simulation and later support after deployment in the experiment.

A firmware project, consisting of a single FPGA, a digital ASIC or a more complex boards with many FPGA’s and additional devices, may be driven by reviews defining milestones at different stages of firmware development but, unless the project is an ASIC, Final or Production Readiness Review do not apply easily, since the development of firmware continues also after deployment of the hardware in the experiment.

Let us assume that the process starts with an Initial Design Review, followed by Progress Reviews at different key points of the firmware development. Two chapters follow, focusing on the Initial Design review and on following Progress Reviews.

Initial Design Review

During this phase, the functionality of the firmware project needs to be defined.

The top level block diagram needs to be described, showing all external interfaces.
If the project is a full board, with more than one FPGA, the top level block diagram is a top level description of the board, with all interfaces and external devices (external to the FPGAs, a memory for example). It will be necessary to describe at least two levels of block diagrams: board-level and

fpga-level, but all that is described below applies in both cases where we have either one or multiple programmable devices.

The document prepared for this review, will also contain top-level parameters:

1. general specifications, for example:
   - timing signal characteristics (examples: L0/L1 max frequency, R3 max frequency)
   - number of input and output links, protocol used, word sizes.
   - which standard interfaces are used for which functionality (examples: GBT, i2c, JTAG, memory)
   - data transfer protocols used (on serial lines for example)
   - max latency of realtime path, if existing
   - where data is stored during L0 and L1(if exists) latencies.
   - max data loss fraction during L0 and L1(if exists) latencies.
   - implications on dead-time settings (for example complex dead-time)
2. parameters that may be mapped directly to HDL (simulation or synthesis) parameters:
   - A. used directly in the design, for example:
     - main clock frequencies,
     - number of ports, IO channels
     - word sizes
   - B. used in simulations, like:
     - timing signal characteristics (examples: L0/L1 max frequency, R3 max frequency)
     - link occupancy
     - packets size
     - expected FE occupancy

Next step is the definition of all main internal blocks, mapping them to the general functionality. In this case we need to define which blocks are of general use overall TDAQ and which ones are specific to this project.
it might be necessary to describe blocks to a lower-level, up to the level where memory blocks appear for example, to define data and control paths.
it is advisable to define all parameters used on each block, so that the good design practice of writing parametrised code becomes visible during the review.
If memory blocks are used and appear at this level, their low level (technology-dependant) interface should be encapsulated, so that only the effectively used signals are appearing in the block description and description becomes technology independent.
Examples of lower-level blocks:

1. blocks of general use (let us call them Common IPs) are:
   - deserialiser blocks (using a specific protocol)
   - timing signals handling blocks
   - initialisation blocks, configuration registers architectural blocks;
   - signal injection blocks
   - standard interfaces (I2C, Memory, IPBus, GLINK)
2. Examples of project specific blocks (let us call them Project IPs) are:
   - custom FE data decoding blocks
   - blocks containing a specific processing algorithm (for example calo clustering, trigger algorithms)

During this step all additional design parameters are defined, like:

- internal bus sizes
- memory sizes
- internal clock speeds

During this step internal clocks are defined together with their clock domains. The clock-domain interfaces have to be described even if their are internal to blocks.

During this phase, the sharing of work among groups will be defined:

1. who takes care of the top-level project assembly and synthesis and of defining top level synthesis constraints.
2. who takes care of designing each individual block for synthesis and developing its basic test bench.
3. who takes care of an high-level description of each individual block, to be used for a verification of the functionality and in more global test-benches.
4. who takes care of simulations for validation:
    1. of each individual block, its test bench (for validation, expecting a yes/no result), also in terms of coverage.
    2. of top level simulations, its test bench (for validation, expecting a yes/no result)
5. who takes care of interfacing the top level with high-level simulation. The test bench should include tests done with realistic FE and data injection, either from Montecarlo techniques (using general parameters like occupancy, L0 rate etcetera) or based on ATLAS simulation, and comparing expected results coming from the high-level description.

An existing model (C/C++) should already exist at this stage to define the functionality of the project. A bit-wise transaction-level (not aware of clock-cycles) model would be preferred.

During this review an estimation of the resource usage of the FPGA (area of the ASIC) should be presented, showing what has been used for an estimation. A list of pin-to-pin compatible devices that are compatible with the FPGA choice should be presented.

During this review a short description of the synthesis constraints used for the timing closure should be described, in order to assess the goodness of the synthesis output.

During this review the code handling methodology should be described (use of git or similar tools for code sharing and versioning, and some ticketing system to handle bugs).

During this review the methodology used for the project, the language(s) used (VHDL, Verilog, SystemC, SystemVerilog, C++, scripting languages) and proprietary tools should be described.

A power estimation should be done at this level, and checked at every progress review.

A description of the control, initialization, monitor and data injection method should be described at this stage.

A description of the testability of the device should be assessed, whether it can be tested stand-alone, if it needs external input or output, where data injection and data sampling is done.

If firmware reviews will be considered for ASIC designs as well, we will need to look at the testability of the project along other lines, like checking the presence of a JTAG chain for board electrical tests, the presence of an internal scan test chain test vectors generation and design coverage of its test vectors, more generally the final test vectors used for device tests at production.

At the end of a successful Initial Design Review, design groups should have a clear picture of what they need to produce, the requirements.
External interfaces and all interfaces on the boundaries of design groups should be defined.
An high level-model might be available to support detailed simulation.

Progress reviews

It is not possible to speak about Final or Pre-production design reviews (unless ASICs are present), but it is possible to monitor the state of progress of a firmware project, checking the status of the design against specific milestones like:

1. external interfaces tests (example: interface link-speed tests), evaluation of performance of single interfaces (for example measurement of serialiser/deserialise latency, working at a given baud rate and using a defined data protocol)
2. single board internal slice test (one-channel in, processing, one-channel out)
3. single-board functionality test
4. system slice-test, where this project is part of a bigger test stand (one-channel in, one-channel out if applies, or a single processing algorithm, out of many, running on all input channels)
5. production test (dedicated firmware, JTAG chain coverage and necessary functional tests where JTAG does not cover parts of the design for example)
6. full system test (after installation)
7. system integration (with detectors and the rest of TDAQ).

More and more of the firmware code will be available moving from a milestone to the next.
In order to progress within each of these tests, the definition and the monitoring of firmware versions, that need to be validated each time against a golden test bench, may be done.

A general review of the methodology can be done at this level, by describing which proprietary and general tools are used during the design, if changed.

Code inspection may be done at this level to check the quality of the synthesis code and the extent/coverage of test benches.

On top of the general good synthesis practice and language usage issues (parametrization, code encapsulation, abstraction from technology choice etc.), that me be verified with code inspection, each project design should be script-based, especially for synthesis and regression tests (yes/no results from validation testbenches).
In this way many steps may be automated without recurring to graphical user interface tools.

During these reviews it has to be made clear at which level simulations and validations are done, behavioural, Register-Transfer, post-synthesis, post-layout.

It needs to be indicated if hand-layout of any blocks is necessary in order to meet the timing closure, and whether the labour-intensive and time consuming tasks like layout work is script-based or done using a graphical user interface.

To understand whether sub-projects may be easily reassigned to groups within the collaboration, it needs to be indicated if any IP used in the project have been purchased, who owns the right to use them.

At the level of single-board test, full system test and system integration test, a description of the DAQ software necessary for configuration, test and monitoring, together with a description of its development and major milestones should be indicated.

Power needs to checked during each review.

Test coverage of all test benches should be evaluated, if they have changed in terms of input data and output checks, or is some more have been added anew.

System-level models and integration with the experiment

It may be useful to develop an environment where a system may be built out of individual “boards”, by describing their connections and each board type, for a full-system simulation, integration and test of our systems (example CMS SWATCH).

Advanced algorithms may nowadays be implemented using programming languages like C, C++ and SystemC, SystemVerilog. Some thoughts might be done on the possibility of writing parts of the code that may be both used for high-level simulations (ATLAS software environment and co-simulation with HDL blocks, usually done using proprietary tools) and synthesis using languages different from HDL (Verilog or VHDL). This approach may simplify the validation/regression tests required by the necessity of having two or more models describing the same functionality (C++ and VHDL for example).

In cases where algorithm menus are built and then synthesised directly, some tools may be developed to automate the generation of synthesis code.

Some thought should be put in supporting, in terms of development tools (like support for emulators) and for final deployment (integration into TDAQ), of the interaction between processing our data part on firmware (synthesised HDL) and part using code running on (embedded) processors. An example may be the Phase-II MDT trigger, which may be partly based on code running on a dedicated ARM processor.

Some thoughts should also go to exploitation of future industry-driven system architectures, which will have embedded processors (ARM) and arrays of GPUs controlled by one or more processors, embedded in the same FPGA containing user logic (an example is the future Xilinx ZYNQ Ultrascale+, with Application processing unit quad-core ARM-A53 @1.5Ghz, with Realtime processing unit dual core ARM-R5 @600 MHz, GPU ARM Mali-400 and programmable logic).

Support of OpenCl should be considered as a standard environment to develop GPU-based applications.

Going further

Further step from this document might be to pick a subset of these ideas and develop them further, to formalise an ATLAS approach to firmware development and management.

This approach is devoted to help designers, share common items and good design practices. A second aim is to allow reviewers to understand the state of development of the firmware part of a larger project, along its many years long timeline.